

Short Paper

Behavior-Based Branch Prediction by Dynamically Clustering Branch Instructions

HANS VANDIERENDONCK, VEERLE DESMET AND KOEN DE BOSSCHERE

Department of Electronics and Information Systems

Ghent University

St.-Pietersnieuwstraat 41

B-9000 Gent, Belgium

E-mail: {hvdieren; vdesmet; kdb}@elis.ugent.be

Conditional branches frequently exhibit similar behavior (bias, time-varying behavior, ...), a property that can be used to improve branch prediction accuracy. Branch clustering constructs groups or clusters of branches with similar behavior and applies different branch prediction techniques to each branch cluster. We revisit the topic of branch clustering with the aim of generalizing branch clustering. We investigate several methods to measure cluster information, with the most effective the storage of information in the branch target buffer. Also, we investigate alternative methods of using the branch cluster identification in the branch predictor. By these improvements we arrive at a branch clustering technique that obtains higher accuracy than previous approaches presented in the literature for the *gshare* predictor. Furthermore, we evaluate our branch clustering technique in a wide range of predictors to show the general applicability of the method. Branch clustering improves the accuracy of the local history (PAG) predictor, the path-based perceptron and the PPM-like predictor, one of the 2004 CBP finalists.

Keywords: microprocessors, speculation, branch prediction, interference, branch clusters

1. INTRODUCTION

Superscalar microprocessors with deep pipelines and high clock frequencies amplify the importance of accurate dynamic branch prediction. This paper researches means to improve the accuracy of branch predictors by correlating with the general behavioral properties of branch instructions.

Many conditional branch instructions behave in a similar way: branches can be strongly biased towards taken or not-taken, they can be correlated with other branches, they can cycle through repeating sequences, *etc.* This paper revisits branch classification: explicitly identifying clusters of branches with similar behavior and exploiting this information to improve branch prediction accuracy.

Branches are labeled with a short branch cluster identification (CID). This CID is made available to the branch predictor. It serves as an additional source of information,

and can be used to index branch prediction tables in a similar way as the program counter and the branch history. Some ways of incorporating the branch CID in the prediction process are illustrated for the gshare [12] predictor (Fig. 1). The predictor can correlate against the CID, just like it correlates against the program counter or the branch history, *e.g.*, by XOR-ing the CID with global history bits (Fig. 1 (b)). A special case of this scheme is partitioning [5, 8]: the CID is concatenated to a hash of the program counter and branch history (Fig. 1 (c)). This has the effect of partitioning the pattern history table (PHT) where each partition of the PHT contains only branches with a specific type of behavior, *e.g.*, branches with a strong taken bias. Partitioning reduces interference in the PHT [8].

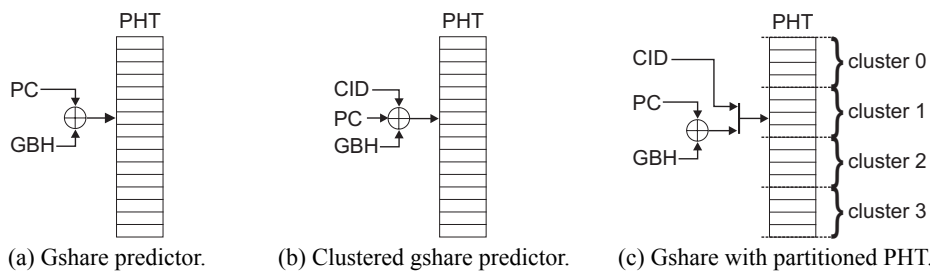


Fig. 1. Clustered branch predictors correlate on an additional source of information: the cluster identification (CID).

In this paper, we revisit the idea of branch clustering with the aim of generalizing the technique. In particular, we investigate new ways to cluster branches in order to obtain higher accuracy than previous approaches. Furthermore, we apply branch clustering to a wide range of recent and highly accurate branch predictors.

The remainder of this paper is organized as follows. Section 2 discusses related work. Schemes to cluster branches based on the taken rate are discussed in section 3 and evaluated in section 4.1. Section 4.2 compares branch clustering to related work and section 4.3 applies branch clustering to various predictor types. Section 5 concludes this paper. A detailed account of this research is available in [16].

2. RELATED WORK

Kim and Tyson [10] analyze the working set characteristics of branch execution. They optimize a PAg (local history) branch predictor based on the observation that the branch predictor only needs to store information for the branches in the working set. As such, problems related to both capacity limitations and interference can be avoided.

In a sense, hybrid branch predictors [3, 12] also cluster branch instructions dynamically. A hybrid branch predictor dynamically selects between either of two component branch predictors. A meta-predictor table learns which predictor is most often correct for each branch instruction. Thus, branches are placed in one of two clusters, depending on which predictor happens to be most accurate.

Chang *et al.* introduce branch classification, a technique to improve the performance of hybrid branch predictors [4]. Branches are classified based on their taken rate. Branches which go the same direction most of the time are best predicted by a short-history predictor, while mixed-direction branches are best predicted by a long-history branch predictor. The presented work differs from Chang *et al.* as we consider other ways of using branch information and we consider different types of predictors.

Chang, Evers and Patt [2] present a dynamic version of branch classification. For global history branch predictors, their technique adds a direction bit and a 3-bit counter to each branch target buffer entry. The counter tracks whether the branch instruction is either always taken or always not-taken over the last 8 branch executions. These branches are predicted using the direction bit. Interference is removed by not updating the PHT for these mostly-one-direction branches. Note that this scheme removes only interference caused by easily predicted branches, which typically do not touch many PHT entries. Their scheme cannot remove interference between mixed-direction branches. This is especially important for local history branch predictors, for which their technique has no effect.

3. CLUSTERING BRANCH INSTRUCTIONS

Branch clusters can be decided in many different ways. The most obvious choice is to consider the taken rate or bias of the branch. The taken rate is the fraction of executions that a branch is taken. Chang *et al.* split off branches that were mostly taken or mostly not-taken [2], *i.e.*, one cluster contains strongly biased branches. The cluster predictor measures the taken rate of a branch and computes the cluster to which the branch belongs. It computes the cluster ID based on the branch address and forwards the cluster ID to the conditional branch predictor (Fig. 2 (a)).

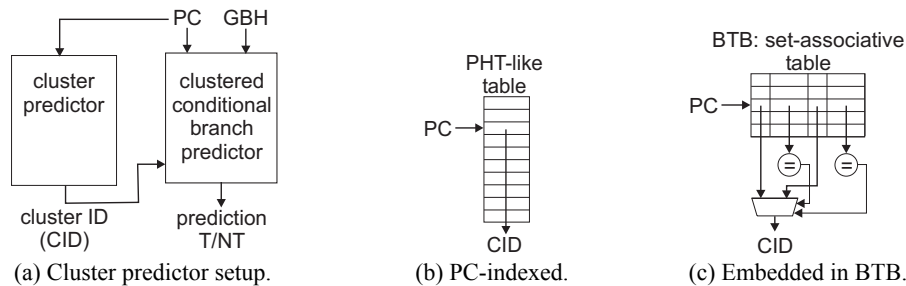


Fig. 2. Combining cluster predictor and branch predictor and the cluster predictors studied in this paper.

The cluster predictor is updated at the same time as the conditional branch predictor. The information stored in the cluster predictor depends on the clustering strategy. For taken rate clustering, it suffices to store a 2-bit saturating up-down counter to track the most recent branch direction. We consider two implementations of dynamic cluster predictors for taken rate clustering (Fig. 2).

3.1 PC-Indexed Cluster Predictor

The PC-indexed cluster predictor consists of a table of 2-bit saturating up-down counters. The table is indexed using low-order bits of the program counter, just like a bimodal predictor (Fig. 2 (b)).

The 2-bit counter read out of the table can be used to cluster the branches into either 2 or 4 clusters, depending on whether clustering is performed only on the branch bias or also on the hysteresis of the counter (weak vs. strong bias).

The PC-indexed scheme has one serious drawback: it suffers from the same destructive interference that it tries to solve. However, this interference should not be problematic in situations where the cluster predictor can be made relatively large. Alternatively, the destructive interference can be avoided by embedding the cluster predictor in the BTB, as explained in the next section.

A number of predictors already contain bimodal components, *e.g.*, tournament-style hybrid branch predictors, the bi-mode predictor [11], YAGS [6] and the geometric history length predictor (GEHL) [14]. In these predictors, the bimodal component can be used to provide the bimodal prediction as well as the cluster identifier for the other components provided that the bimodal component is updated for all predictions.

3.2 Cluster Predictor Embedded in the BTB

The cluster predictor can also be embedded in the branch target buffer (BTB). The idea is to leverage the associativity and the size of the BTB to improve the accuracy of the cluster predictor (Fig. 2 (c)). Indeed the size (*e.g.*, 4096 branches in the Pentium 4 [7]) and the presence of associativity remove interference from the cluster predictor.

It is necessary to make an intelligent guess of the proper cluster in the case of a BTB miss. A BTB miss either means that the BTB suffers from capacity misses or that a branch is never taken. The latter type of branches is never inserted in the BTB to save space. Thus, we predict a not-taken cluster for BTB misses.

4. EVALUATION

The effectiveness of branch clustering is evaluated using the SPECint2000 benchmarks¹ executing the reference inputs. The C benchmarks are compiled for the Alpha ISA with the Compaq C compiler with optimization flags `-arch ev6 -fast`. Eon is similarly compiled using the Compaq C++ compiler. We perform trace-based simulation of the benchmarks. The traces are constructed using a functional simulator from the Simple-Scalar Tool Set [1]. The first 50 million conditional branches of each benchmark, corresponding to approximately 300 million instructions, are skipped. Branch prediction accuracy is measured over the following 250 million conditional branches. This corresponds roughly to 1.5 billion instructions. Accuracy is summarized over all benchmarks using the arithmetic mean.

We use the conditional branch misprediction rate as performance measure. The cost

¹ See <http://www.spec.org/cpu2000/>.

of each prediction scheme is modeled in terms of the storage budget. The BTB configuration is fixed to contain 1024 branches in a 4-way set-associative configuration. This BTB is sufficiently large for our benchmarks. The additional cost for clustering equals the bits necessary to store branch cluster information. Hereby, the relative storage size increase of the BTB should be within 5%.

4.1 Cluster Predictors

We compare cluster predictors by applying them to partition a gshare branch predictor [12] because branch cluster information is not useful or correct by itself. The utility of the branch cluster information is evaluated by means of the gshare misprediction rate.

4.1.1 PC-indexed cluster predictor

Fig. 3 shows the misprediction rate of a 4-way partitioned gshare with various sizes of the cluster predictor. Larger cluster predictors are more effective than smaller ones. Thus, the PC-indexed cluster predictor is sensitive to interference, just like a bimodal branch predictor. However, starting at 1KB, the reduction in misprediction rate does not weigh up compared to the increase in storage budget.

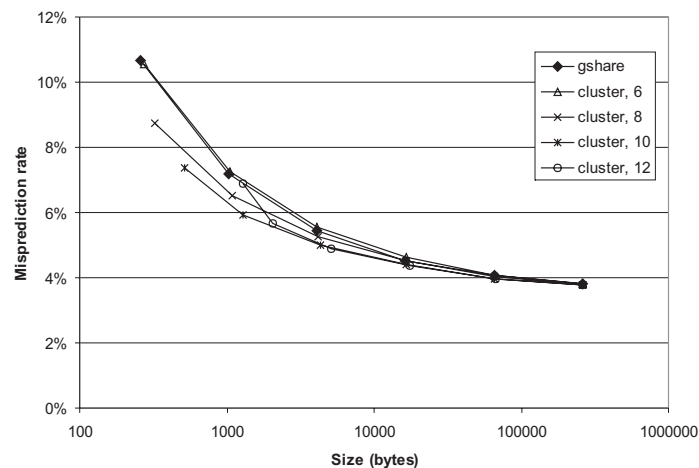


Fig. 3. The gshare predictor partitioned in 4 clusters. The clusters are determined by means of a bimodal PC-indexed table. The number in the index shows the base-2 logarithm of the number of entries in the cluster predictor.

Note that an interesting side effect of having a cluster predictor is that it becomes possible to construct predictors with storage budgets that are not a power of 2. When trading off size versus accuracy of a gshare predictor, then every increase in size equals doubling the size. With the clustered gshare predictor, one has finer control over size increments. Additional configurations exist that require less storage and are more accurate than a two-times larger gshare.

4.1.2 Cluster prediction with a BTB

By storing the cluster ID in the BTB we can avoid interference effects existent in the PC-indexed scheme. This bears no additional cost over the PC-indexed scheme, because the BTB is already a tagged set-associative structure. Partitioning a 1KB gshare predictor with the BTB reduces the misprediction rate from 7.2% to 5.7% in the case of two clusters and to 5.8% in the case of four clusters (Fig. 4). This constitutes a 20% reduction in mispredictions. At 8KB, the misprediction rate is reduced from 5.0% to 4.6% and 4.7% for 2-way and 4-way partitioning, respectively.

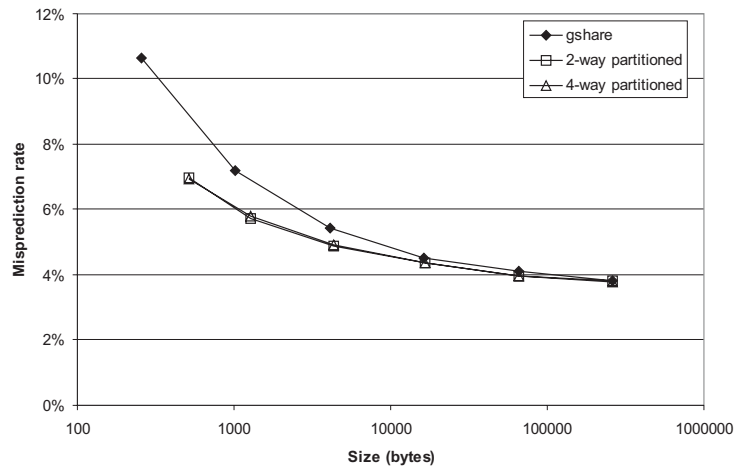


Fig. 4. Clustering the gshare predictor with cluster identification by the BTB.

4.1.3 Comparison of cluster predictors and conclusion

We compare cluster predictors applied to partitioning the gshare predictor in 4 partitions. The PC-indexed and BTB cluster predictors have the same storage cost, *i.e.*, they contain information on the same number of branches. BTB cluster prediction is slightly more accurate than PC-indexed cluster prediction. The 1KB gshare has a 6.0% misprediction rate with PC-indexed clustering and a 5.8% misprediction rate with BTB clustering. At 8KB, the misprediction rates are 4.6% and 4.7%, respectively.

Partitioning makes gshare more accurate for all benchmarks at a storage budget of 1KB (Fig. 5). In some unusual circumstances, partitioning increases the misprediction rate, *e.g.*, for eon in the 8KB predictor. It is hard to explain this phenomenon, as it does not occur for eon in the 4KB, nor the 16KB predictors.

For all cluster predictors, we found little difference in partitioning gshare in 2 or 4 clusters. Smaller predictors are slightly better with 4 clusters and large predictors are better off with 2 clusters.

We also made a comparison of dynamically assigned branch clusters to statically assigned branch clusters, based on profile information. The statically assigned branch clusters yield a better trade-off between prediction accuracy and predictor size, mainly

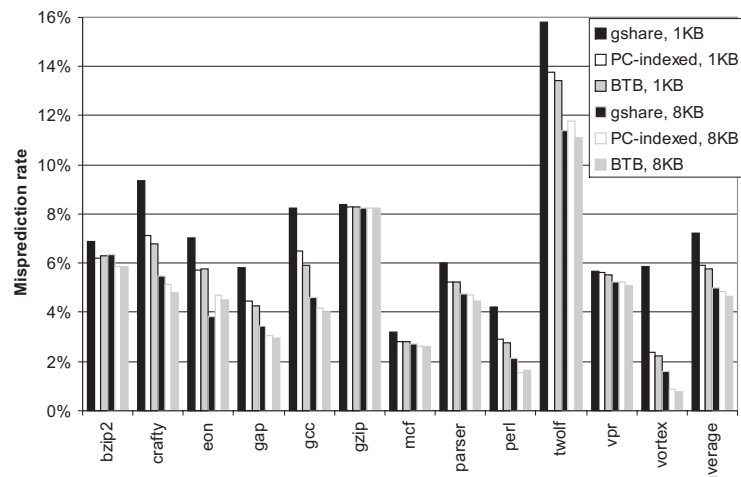


Fig. 5. Per-benchmark results for 1KB and 8KB gshare predictors.

because no storage space is required to learn the branch clusters. Profile-based branch cluster assignment may be beneficial provided that the CIDs can be made available to the architecture without an associated cost [16].

4.2 Comparison to Related Techniques

The literature describes many schemes to eliminate interference from branch prediction tables. We compare against two particular techniques: the agree predictor and branch classification.

4.2.1 The agree predictor

The agree predictor [15] reduces interference by modifying the *meaning* of the PHT contents: instead of storing the branch direction, the PHT stores whether a branch *agrees* with the branch bias or not. The branch bias is predicted by a bias predictor. It was proposed to store a single bit in a tagged table indicating the branch direction [15]. The bit is initialized upon a miss in the table. It is never updated afterwards. To make fair comparisons to branch clustering, we store the bias bit in the BTB. Branches missing in the BTB are predicted to have a not-taken bias.

The agree predictor outperforms the gshare predictor (Fig. 6), although there is a slight degradation for very large predictor sizes. Clustering gshare in 2 partitions outperforms the agree predictor by far. For a 1KB predictor, the agree predictor has a 6.7% misprediction rate and the partitioned gshare predictor has a 5.7% misprediction rate, resulting in a reduction of 11%.

4.2.2 Branch classification

Branch classification [2] is similar to branch clustering, as this technique tries to separate mostly-one-direction branches from the others. The application presented in [2]

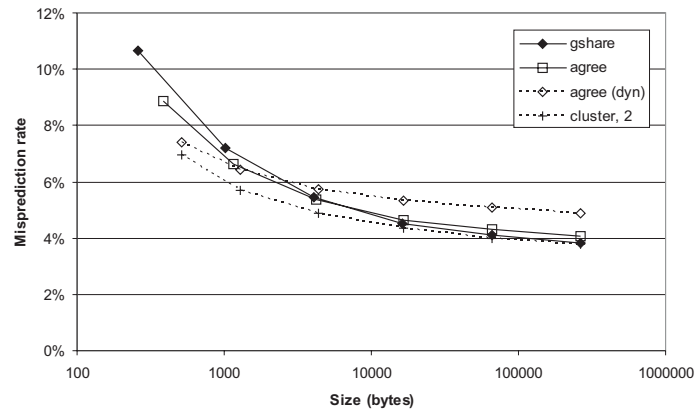


Fig. 6. Comparison to the agree predictor.

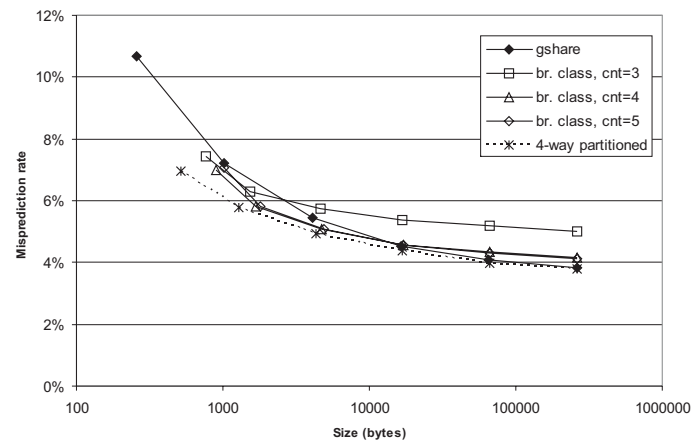


Fig. 7. Comparison to branch classification for a gshare predictor.

is to use a static prediction for mostly-one-direction branches. This application is, however, not as successful as the applications presented in this paper. Clusters are predicted by adding a direction bit and a counter to each entry of the BTB. The direction provides the static prediction and the counter tracks how many consecutive occurrences of the branch went the same direction. When the counter is saturated, the branch is predicted using the direction bit and the PHT is not updated.

This implementation of branch classification is more costly than branch clustering, as the counter should be 3 bits wide [2]. We compare branch classification to branch clustering using 2-bit saturating counters to track the branch direction (Fig. 7). This shows that branch classification is less accurate than branch clustering. Branch classification has a misprediction rate of 7% while partitioning results in a misprediction rate of 5.7%, an improvement of 17.5%. Furthermore, for very large branch predictors, branch classification is less accurate than the original gshare predictor. For these large, and accurate, predictors, the static prediction for strongly biased branches is less accurate than

gshare itself. Therefore, we increased the size of the counter to 4 and 5 bits, such that the static predictor is used for fewer branches. This modification helps branch classification, but it remains less accurate than the baseline gshare.

4.3 Application to Various Predictors

We apply clustering to the PAg (local history) predictor [17], a path-based perceptron predictor [9] and the PPM-like predictor [13]. This section uses the BTB cluster predictor. Aside to these straightforward applications, clustering allows many other optimizations related to tuning the per-cluster predictors to the specific per-cluster behavioral properties (*e.g.*, predictor size, history length, ...). Space limitations however, prohibit us from going into details.

4.3.1 The local-history (PAg) predictor

For the PAg predictor, we vary the BHT size from 16 ($s = 4$) to 1024 ($s = 10$) entries while local history lengths are 8, 10, 12, 14 or 16 (Fig. 8). The PAg can be improved by either partitioning the branch history table (BHT), the pattern history table (PHT), or both. All choices lead to similar results, although partitioning the PHT only is most accurate. At 1KB clustering drops the misprediction rate from 8.5% ($s = 8$) down to 7.1% ($s = 6$), removing 16% of the mispredictions. Note also that the partitioned PAg predictors all have accuracies in the same range as the baseline PAg with 1024 entries in the BHT. In other words, partitioning the PAg predictor reduces the need for a large BHT.

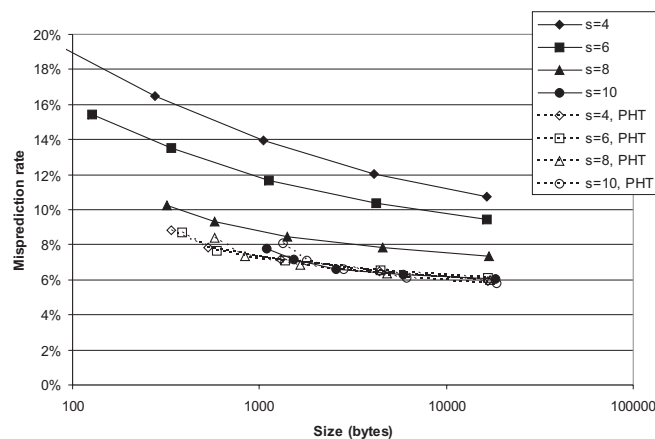
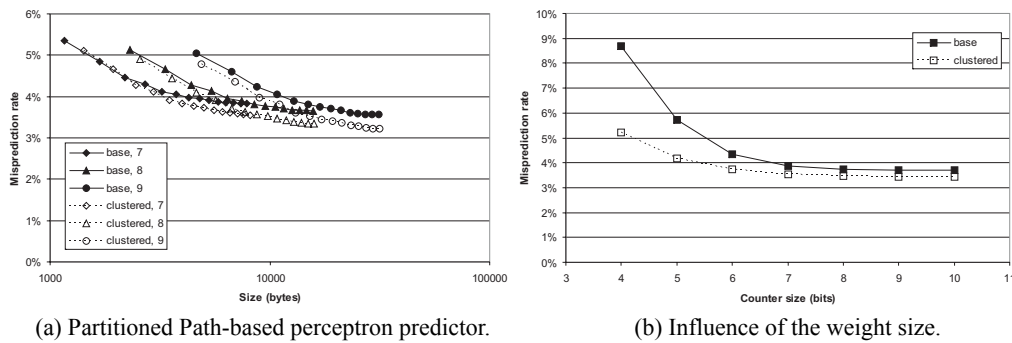


Fig. 8. Clustering the PAg predictor.

4.3.2 Path-based perceptron predictor

We partition a path-based perceptron predictor [9] with 7, 8 or 9-bit index in the prediction table and history length varying from 8 to 60 global branch history bits in steps of 4 bits. The perceptron weights are 8 bits wide and the update threshold is computed using the formula $20.58 + 2.14(h + 1)$, where h is the history length [9]. The



(a) Partitioned Path-based perceptron predictor.

(b) Influence of the weight size.

Fig. 9. Partitioning the path-based perceptron predictor.

partitioned predictor with 7-bit index is between 4.4% (shortest history) and 7.5% (longest history) more accurate than the baseline (Fig. 9 (a)). The partitioned predictor with 9-bit index sees improvements ranging from 5.5% to 9.7%.

Part of the interference in a perceptron predictor is eliminated by increasing weight size. Conversely, when aliasing is reduced, then weight size may also be reduced. The mechanism works as follows. To learn a branch outcome, the weighted sum of previous branch outcomes must exceed the update threshold. However, when the weight for a history bit is aliased, *i.e.*, some program counters require a positive weight while others require a negative weight, then the weight is trained to a value close to zero. Thus, aliased weights do not contribute to the sum of weights. To make the sum of weights exceed the update threshold, the unaliased weights must take on larger values. To accommodate large values the maximum weight size and corresponding storage budget must be increased when aliasing is frequent.

Clustering the path-based predictor eliminates interference, so the size of the weights has a lower impact on accuracy. In the baseline predictor with 256 perceptrons and history length 40, the misprediction rate increases by 3.2% when decreasing the weight size from 8 to 7 bits (Fig. 9 (b)). It increases by only 1.9% in the same-sized clustered predictor. Furthermore, the clustered predictor with 7-bit weights remains 5.5% more accurate than the non-clustered predictor with 8-bit weights (3.5% vs. 3.7%).

The above observation allows trading-off weight size versus history length. A clustered path-based perceptron predictor with 256 predictors, history length 21 and weight size of 8 bits has an average misprediction rate of 3.8%. When reducing weight size to 7 bits and increasing history length to 24 then the misprediction rate becomes 3.7% for the same storage budget.

4.3.3 The PPM-like predictor

A final application is the PPM-like predictor [13]. Our experiments assume that all tables have the same number of entries. Other parameters are chosen identical to [13]. Partitioning all tables improves the misprediction rate from 4.7% to 4.4% at 2KB. However, partitioning is not beneficial at larger budgets. Better results are obtained by partitioning only the bimodal table, reducing the misprediction rate to 4.2% at 2KB, a 10% improvement.

5. CONCLUSION

This paper revisits branch clustering: exploiting behavioral properties of branch instructions to increase branch prediction accuracy. Branches with similar behavioral properties are placed in the same branch cluster. The branch cluster information is fed into the branch predictor to increase its prediction accuracy.

In this paper we analyze two methods of clustering based on the taken rate: using a PC-indexed table and storing cluster prediction information in the BTB. These cluster predictors improve the branch accuracy of a partitioned gshare predictor by 3.2% for the PC-indexed scheme and 6.3% for the BTB scheme. For a 1KB gshare predictor, the BTB scheme is 11% more accurate than branch classification and 17.5% more accurate than the agree predictor.

We also show the utility of branch clustering for other predictors. In the PAg (local history) predictor, partitioning the PHT reduces the need for a large BHT and improves the accuracy of a 1KB PAg by 16%. Partitioning an 8KB path-based perceptron predictor increases accuracy by 7.5% to 9.7% depending on the table size. Finally, partitioning the bimodal table of a 2KB PPM-like reduces the misprediction rate with 10%.

ACKNOWLEDGEMENTS

Hans Vandierendonck is a Post-doctoral Fellow with the Fund for Scientific Research-Flanders (FWO-Vlaanderen). This research is partially sponsored by the Flemish Institute for the Promotion of Scientific-Technological Research in the Industry (IWT), the HIPEAC European Network of Excellence and Ghent University.

REFERENCES

1. D. Burger, T. M. Austin, and S. Bennett, "Evaluating future microprocessors: the SimpleScalar tool set," Technical report, Computer Sciences Department, University of Wisconsin, Madison, 1996.
2. P. Y. Chang, M. Evers, and Y. Patt, "Improving branch prediction accuracy by reducing pattern history table interference," in *Proceedings of Conference on Parallel Architectures and Compilation Techniques*, 1996, pp. 48-57.
3. P. Y. Chang, E. Hao, and Y. Patt, "Alternative implementations of hybrid branch predictors" in *Proceedings of the 28th Annual International Symposium on Microarchitecture*, 1995, pp. 252-257.
4. P. Y. Chang, E. Hao, T. Y. Yeh, and Y. Patt, "Branch classification: a new mechanism for improving branch predictor performance," in *Proceedings of the 27th Annual International Symposium on Microarchitecture*, 1994, pp. 22-31.
5. C. M. Chen and C. T. King, "Walk-time address adjustment for improving the accuracy of dynamic branch prediction," *IEEE Transactions on Computers*, Vol. 48, 1999, pp. 457-469.
6. A. Eden and T. Mudge, "The YAGS branch prediction scheme," in *Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture*, 1998, pp. 69-77.

7. Intel Corporation, *IA-32 Intel Architecture Software Developers Manual Volume 1: Basic Architecture*, 2001.
8. D. Jiménez, “Code placement for improving dynamic branch prediction accuracy,” in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2005, pp. 107-116.
9. D. Jiménez and C. Lin, “Fast path-based neural branch prediction,” in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, 2003, pp. 243-252.
10. S. Kim and G. Tyson, “Analyzing the working set characteristics of branch execution,” in *Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture*, 1998, pp. 49-58.
11. C. C. Lee, I. C. Chen, and T. Mudge, “The bimode branch predictor,” in *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*, 1997, pp. 4-13.
12. S. McFarling, “Combining branch predictors,” Technical Report No. WRL TN-36, Western Research Laboratory, Digital Equipment Corporation, 1993.
13. M. Michaud, “A PPM-like, tag-based predictor,” *The Journal of Instruction-Level Parallellism*, <http://www.jilp.org/vol7/>, Vol. 7, 2005.
14. A. Sez nec, “Analysis of the O-GEometric history length branch predictor,” in *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, 2005, pp. 394-405.
15. E. Sprangle, R. S. Chappell, M. Alsup, and Y. N. Patt, “The agree predictor: a mechanism for reducing negative branch history interference,” in *Proceedings of the 24th Annual International Symposium on Computer Architecture*, 1997, pp. 284-291.
16. H. Vandierendonck, V. Desmet, and K. de Bosschere, “Behavior-based branch prediction by dynamically clustering branch instructions,” Technical Report, Department of Electronics and Information Systems, Ghent University, 2006.
17. T. Y. Yeh and Y. Patt, “A comparison of dynamic branch predictors that use two levels of branch history,” in *Proceedings of the 20th Annual International Symposium on Computer Architecture*, 1993, pp. 257-266.

Hans Vandierendonck is a post-doctoral research fellow with the Fund for Scientific Research-Flanders (FWO-Flanders), located at the Faculty of Applied Sciences of Ghent University. He graduated in Engineering from Ghent University in 1999 and received the Ph.D. degree from the same university in 2004. He is a member of the Association for Computing Machinery and of the Institute of Electrical and Electronics Engineers. His research interests are focused in the area of computer architecture, with an emphasis on memory hierarchy optimizations, branch prediction, value prediction and the characterization and generation of benchmarks.

Veerle Desmet was born in Ghent, Belgium in 1978. She received the Engineering degree and Ph.D. degree in Computer Science from Ghent University, Belgium, in 2001 and 2006, respectively. Veerle Desmet currently is a postdoctoral researcher at the De-

partment of Electronics and Information Systems (ELIS) at the same university. The topic of her research is in the area of computer architecture, more specific on branch prediction. She was a finalist in the 1st Journal of Instruction-Level Parallelism Championship Branch Prediction in 2004.

Koen de Bosschere is professor at the Engineering Faculty of the Ghent University where he teaches courses on Computer Architecture and Operating Systems. His current research interests are global optimization, performance modelling, microarchitecture, and debugging.